**IN THE SPECIFICATION:**

Please replace paragraph [0001] of the original specification with the following amended paragraph:

**[0001]** The present invention relates generally to computing systems[[ ]], and more particularly, to techniques for processing events for concurrent tasks in a virtual machine.

Please replace paragraph [0002] of the original specification with the following amended paragraph:

**[0002]** One of the goals of high-level computer programming languages is to provide a portable programming environment. In such an environment, computer programs may easily be ported to another computer platform. High-level languages such as "C" provide a level of abstraction from the underlying computer architecture and their success is well evidenced from the fact that most computer applications are now written in a high level language.

Please replace paragraph [0003] of the original specification with the following amended paragraph:

**[0003]** Portability has been taken to new heights with the advent of the World Wide Web ("the Web") as an interface protocol for the Internet. The World Wide Web allows communication between diverse computer platforms through a graphical interface. Computers communicating over the Web are able to download and execute small applications called applets. Given that applets may be executed on a diverse assortment of computer platforms, the applets are typically executed by a Java™ virtual machine. In general, a virtual machine is an abstract computing machine that is implemented over [[a]] specific hardware and/or software. The virtual machine can support platform applications for the specific hardware and/or software. Given that the

Java™ virtual machine is more prevalent, the Java computing environment will be discussed further.

Please replace paragraph [0008] of the original specification with the following amended paragraph:

**[0008]** Virtual machines are highly useful. However, as will be discussed below, conventional virtual machines do not provide a multi-tasking environment in a manner that would allow a user to concurrently perform some tasks. These tasks, for example, include interactive tasks (e.g., application programs) that receive input ~~form~~ from the user or another source.

Please replace paragraph [0009] of the original specification with the following amended paragraph:

**[0009]** In a multi-tasking computing environment, a task can be referred to as a single computing unit that typically performs a task. The computing unit can, for example, be represented by a set of programming instructions (e.g., high-level programming instructions written in C, C++, or Java). On the other hand, a task may also be an entire application program (or applet). By way of example, a task may be a calculator, a music player, a video player, or an interactive game. The user can initiate and use each [[on]] of these tasks to form a function or computing task.

Please replace paragraph [0010] of the original specification with the following amended paragraph:

**[0010]** In any case, a computing system that supports multi-tasking allows tasks to run concurrently. Concurrent tasks typically take turns to execute and/or share the resources of the computer system. However, from the user's perspective multiple tasks can be performed

concurrently. By way of example, a user can use a calculator application and a music player concurrently. This allows the user to do calculations while listening to the music. Conventional virtual machines, however, do not provide a multi-tasking environment in a manner that would allow a user to concurrently use tasks that are interactive. This means that the user of a conventional virtual machine cannot effectively interact with two concurrent tasks.

Please replace paragraph [0012] of the original specification with the following amended paragraph:

**[0012]** As noted above, virtual machines are highly useful in today's computing environments. Furthermore, it is desirable to develop virtual machines that support multi-tasking because such an environment would also allow users to perform multiple tasks concurrently.

Please replace paragraph [0013] of the original specification with the following amended paragraph:

**[0013]** In a multi-tasking computing environment, a task can be referred to as a unit of computing that typically performs a task. The computing unit can, for example, be represented by a set of programming instructions (e.g., high-level programming instruction written in C, C++, or Java). A task may also be or can be associated with an entire application program, or applet. By way of example, a task may be a calculator, a music player, a video player, a computer game. In any case, a computing environment that supports multi-tasking allows tasks to run concurrently. These multiple tasks can take turn to execute and/or share the resources of a computer system. This allows a user to perform multiple tasks. By way of example, a user can use a calculator application and a music player concurrently. This allows the user to do calculations while listening to the music.

Please replace paragraph [0014] of the original specification with the following amended paragraph:

**[0014]** Some tasks, however, require "event" processing. These tasks, for example, include interactive programs that receive input from users, and as such are typically more interesting to the users. Event processing includes delivering and handling external events (e.g., input received from the user from a keyboard) to the appropriate task (e.g. an interactive game). In [[a]] virtual machines, external events are typically generated, transmitted, or processed by platform specific hardware or software components. These platform-specific events can be referred to as "native" events. Processing native events in a virtual machine introduces new problems that are not typically addressed in more conventional computing environments where application programs are platform-dependent. As such, processing native events in a virtual machine need to be addressed.

Please replace paragraph [0020] of the original specification with the following amended paragraph:

**[0020]** As noted in the background section, virtual machines are highly useful in today's computing environments. Furthermore, it is desirable to develop virtual machines that support multi-tasking because such environment would also allow users to perform multiple tasks concurrently.

Please replace paragraph [0021] of the original specification with the following amended paragraph:

**[0021]** In a multi-tasking computing environment, a task can be referred to a unit of computing that typically performs a task. The computing unit can, for example, be represented by a set of programming instructions (e.g., high-level programming instruction written in C, C++, or Java).

A task may also be or can be associated with an entire application program or applet. By way of example, a task may be a calculator, a music player, a video player, or a computer game. In any case, a computing environment that supports multi-tasking allows tasks to run concurrently. These multiple tasks can take turn to execute and/or share the resources of a computer system. This allows a user to perform multiple tasks. By way of example, a user can use a calculator application and a music player concurrently. This allows the user to do calculations while listening to the music.

Please replace paragraph [0022] of the original specification with the following amended paragraph:

**[0022]** Processing some tasks, however, require "event" processing. These tasks, for example, include interactive programs that receive input from users, and as such are typically more interesting to the users. Event processing includes delivering and handling external events (e.g., input received from the user from a keyboard) to the appropriate task (e.g. an interactive game). In [[a]] virtual machines, external events are typically generated, transmitted, or processed by platform specific hardware or software components. These platform-specific events can be referred to as "native" events. Processing native events in a virtual machine introduces new problems that are not typically addressed in more conventional computing environments where application programs are platform-dependent. As such, processing native events in a virtual machine need to be addressed.

Please replace paragraph [0026] of the original specification with the following amended paragraph:

**[0026]** FIG. 1B illustrates a computing environment 110 in accordance with one embodiment of the invention. As shown in FIG. 1B, a virtual machine 112 supports an application layer 114

over a platform 116. The application layer 114 represents one or more applications (e.g., applets) that are supported by the virtual machine 112. The platform 116 represents hardware and/or software specific components of the computing environment 110. It should be noted that the one or more applications in the application layer 114 that are supported by the virtual machine 112, can be platform-independent. In other words, an application in the application layer 114 may be ported to a different platform (~~now~~ not shown). It should also be noted that the virtual machine 112 supports two concurrent tasks, namely task 120 and task 122 that are operating concurrently on virtual machine 112.

Please replace paragraph [0027] of the original specification with the following amended paragraph:

**[0027]** As illustrated in FIG. 1B, the virtual machine 112 receives several platform specific events (or native events) E1, E2, [[E2]] E3 and E4 that are associated with tasks 120 and 122. These native events can, for example, be incoming data received via a network device (or Interface), input generated by a user via a keyboard, and so on. In general, a native event can be any event that is a generated, processed, or transmitted by the platform 116. In any case, these native events are to be presented to appropriate task for processing by the virtual machine 112.

Please replace paragraph [0029] of the original specification with the following amended paragraph:

**[0029]** The determination made by the smart event-dispatcher 118 can be made based on a variety of desired criteria. By way of example, native events generated or received from a particular entity (e.g., device) can be assigned to a particular task. In another embodiment, native events are dispatched to the task that is currently in the foreground (e.g., currently being displayed for the user). The native event dispatcher 118 can also, for example, dispatch events

based on user selection (e.g., be programmed by the user), or dispatch events based on event attributes (e.g., content, incoming, port, proximity, etc.) In any case, the smart event-dispatcher [[108]] 118 delivers a native event to a particular task in the multi-tasking environments supported by the virtual machine [[102]] 112.

Please replace paragraph [0030] of the original specification with the following amended paragraph:

**[0030]** FIG. 1C illustrates a method 150 for processing[[,]] a platform-dependent (native) event associated with a specific platform[[,]] in a virtual machine that supports concurrent tasks in accordance with one embodiment of the invention. The method 150 can, for example, be used by the virtual machine 112 to process native events associated with the platform ~~106~~ 116 (shown in FIG. 1B). Initially, at operation 152, a platform-dependent (native) event associated with a specific platform is received. Next, at operation 154, it is determined which task that has been assigned to process the platform-dependent (native) event. Thereafter, the method 150 proceeds to operation 156 where the platform-dependent (native) event is ~~send~~ sent to the task assigned to process the platform-dependent (native) event. Accordingly, at operation 158, the platform-dependent (native) event is processed by the task that has been assigned to process it. The method 150 ends following operation 158.

Please replace paragraph [0033] of the original specification with the following amended paragraph:

[0033] As such, a smart event-dispatcher (or manager) 212 can dispatch a platform-specific native event E1 to a task 214 while another event E2 is dispatched to another task 214. Again, it should be noted that tasks 214 and 216 are concurrently supported by the virtual machine 204. In this embodiment, the smart event-dispatcher (or manager) 212 is implemented as an event-

manager thread the waits on native events and dispatches them to various tasks. As shown in FIG. 2, the smart event-dispatcher (or manager) 212 includes wait-on-event 216 and event-dispatching logic 218. In effect, the wait-on-event 216 causes the smart event dispatcher (or manager) 212 to wait until a native event is received. When, for example, a native event E2 is received, the event-dispatching logic 218 is activated. The event-dispatching logic 218 determines which task should receive the native event E2. As noted above, this determination can be made based a variety of different criteria. In any case, the smart event-dispatcher (or manager) 212 delivers the native event E2 to at least one of a plurality of tasks (tasks 214 and ~~216~~ 217) based on the event-dispatching logic 218.

Please replace paragraph [0034] of the original specification with the following amended paragraph:

**[0034]** As illustrated in FIG. 2, the smart event dispatcher (or manager) 212 delivers the native event E2 to the task ~~216~~ 217 . More particularly, the smart event-dispatcher (or manager) 212 places the native event E2 in an event-repository (e.g., First-in First-out queue) 220. The smart event-dispatcher (or manager) 212 also notifies an event-handler 221 associated with task ~~216~~ 217 that a native event has been delivered. This causes the event-handler 221 (e.g., an event handler thread) associated with task ~~216~~ 217 to be invoked. In effect, the notification of the delivery causes a wait-on-event logic 222 to be invoked. As a result, an event-processing logic 223 can access the event-repository 220 and process event E2 on behalf of the task ~~216~~ 217. It should be noted that the smart event-dispatcher (or manager) 212 can manipulate data (e.g., encapsulate it) before placing it in the event-repository 220. This operation is further illustrated below (see, for example, operation 330 of FIG. 3).

Please replace paragraph [0035] of the original specification with the following amended paragraph:

**[0035]** It should be noted that tasks ~~216~~ 214 and ~~218~~ 217 can, for example, be associated with Mobile Information Device Profile (MIDP) applications 224 and 226 (or midlets) that are complaint with Java Mobile Information Device Profile 228. The Mobile Information Device Profile 228 is described in Java Specification for Mobile Information Device Profile (JSR-37) which has been incorporated by references for all purposes.

Please replace paragraph [0036] of the original specification with the following amended paragraph:

**[0036]** It should also be noted that the techniques for events dispatching illustrated in FIG. 2 can also be implemented in computing environments that are not complaint with Java Specification for Mobile Information Device Profile (JSR-37). This should be evident because, among other things, the event manager, event handler, and event queue can be implemented in any virtual machine to dispatch events in a similar manner as discussed above. As such, these techniques can be used to dispatch events in any virtual machine.

Please replace paragraph [0040] of the original specification with the following amended paragraph:

**[0040]** The processing 320, however, starts at operation 322 where a determination is made as to whether a native event has been received. The processing 320 waits until operation 322 determines that a native event has been received. If it is determined at operation 322 that a native event has been received, the processing 320 proceeds to operation 324 where the task-ID of the foreground task is retrieved. Thereafter, at operation 326, a Java object (e.g., MIDlet object)

associated with the task identified by the task-ID is obtained. This Java object is then used to get handles (e.g., reference) on the event-repository and an event-handler for the foreground task. After[[,]] the handles for the event-repository and an event-handler are obtained, the native event is encapsulated at operation 328 so that it can be represented as a Java event object (e.g., MIDlet event object). Accordingly, at operation 330, the Java event object is placed in the event-repository using the handle of the event-repository of the foreground task. Finally, at operation 332, the event-handler can be notified using its handle [[ ]]. This can, for example, be achieved by sending a "thread.notify" to an event handler thread. In any case, the notification alerts the event-handler that a Java event object has been queued in the event-repository. The processing 320 then proceeds to operation 320 ~~where~~ when it is determined whether a native event has been received.

Please replace paragraph [0041] of the original specification with the following amended paragraph:

**[0041]** As will be appreciated by those skilled in the art, some tasks implemented in Java may not be able to access data in data structures that are not implemented in Java (i.e., a native programming language). One such example is Java tasks that do not share data with each other or tasks that are in other ways isolated from each other. These tasks can be referred to as "isolates." Isolates and possibly other Java tasks may not be able to directly access event repository and handlers that are represented as data structures that are implemented in Java.